GeeksforGeeks

Pandas

```
import numpy as np

import pandas as pd
```

`Table of Contents`

1. Working with Pandas Series

`a) Creating Series`

Pandas Series is a one-dimensional labeled array capable of holding data of any type (integer, string, float, python objects, etc.). The axis labels are collectively called index. Labels need not be unique but must be a hashable type. The object supports both integer and label-based indexing and provides a host of methods for performing operations involving the index.

Seires through list

```
lst = [1,2,3,4,5]

pd.Series(lst)
```

```
0    1
1    2
2    3
3    4
4    5
dtype: int64
```

Series through Numpy array

```
arr = np.array([1,2,3,4,5])
print(pd.Series(arr))
print('*'*50)
arr1=np.array([1,2,3,4,5])
print(pd.Series(arr1))
```

```
0    1
1    2
2    3
3    4
4    5
dtype: int64
**************************************************
0    1
1    2
2    3
3    4
4    5
dtype: int64
```

Giving Index from our own end

```
pd.Series(index = ['Eshant', 'Pranjal', 'Jayesh', 'Ashish'], data =
[1,2,3,4])
```

```
Eshant     1
Pranjal    2
Jayesh     3
Ashish     4
dtype: int64
```

Series through Dictionary values.

```
steps = {'day1' : 4000, 'day2' : 3000, 'day3' : 12000}
pd.Series(steps)
```

```
day1     4000
day2     3000
day3    12000
dtype: int64
```

Using `repeat` function along with creating a Series

Pandas Series.repeat() function repeat elements of a Series. It returns a new Series where each element of the current Series is repeated consecutively a given number of times.

```
pd.Series(5).repeat(3)
```

```
0    5
0    5
0    5
dtype: int64
```

we can use the reset function to make the index accurate

```
pd.Series(5).repeat(3).reset_index(drop = True)
```

```
0    5
1    5
2    5
dtype: int64
```

This code indicates:

- 10 should be repeated 5 times and
- 20 should be repeated 2 times

```
s = pd.Series([10,20]).repeat([5,2]).reset_index(drop = True)

s
```

```
0    10
1    10
2    10
3    10
4    10
5    20
6    20
dtype: int64
```

Accessing elements

```
s[4]
```

```
np.int64(10)
```

s[0] or s[50] something like this would not work becasue the we can access elements based on the index which we procided

```
s[6]
```

```
np.int64(20)
```

By last n numbers (start - end-1)

```
s[2:-2]
```

```
2    10
3    10
4    10
dtype: int64
```

`b) Aggregate function on pandas Series`

Pandas Series.aggregate() function aggregate using one or more operations over the specified axis in the given series object.

```
sr = pd.Series([1,2,3,4,5,6,7])

sr.agg([min,max,sum])
```

```
C:\Users\afroz\AppData\Local\Temp\ipykernel_2244\3916131715.py:3:
FutureWarning: The provided callable <built-in function min> is
currently using Series.min. In a future version of pandas, the
provided callable will be used directly. To keep current behavior pass
the string "min" instead.
  sr.agg([min,max,sum])
C:\Users\afroz\AppData\Local\Temp\ipykernel_2244\3916131715.py:3:
FutureWarning: The provided callable <built-in function max> is
currently using Series.max. In a future version of pandas, the
provided callable will be used directly. To keep current behavior pass
the string "max" instead.
  sr.agg([min,max,sum])
C:\Users\afroz\AppData\Local\Temp\ipykernel_2244\3916131715.py:3:
FutureWarning: The provided callable <built-in function sum> is
currently using Series.sum. In a future version of pandas, the
provided callable will be used directly. To keep current behavior pass
the string "sum" instead.
  sr.agg([min,max,sum])
```

```
min       1
max       7
sum      28
dtype: int64
```

`c) Series absolute function`

Pandas Series.abs() method is used to get the absolute numeric value of each element in Series/DataFrame.

```
sr = pd.Series([1,-2,3,-4,5,-6,7])

sr.abs()
```

```
0    1
1    2
2    3
3    4
4    5
5    6
6    7
dtype: int64
```

`d) Appending Series`

Pandas Series.append() function is used to concatenate two or more series object.

Syntax: Series.append(to_append, ignore_index=False, verify_integrity=False)

Parameter : to_append : Series or list/tuple of Series ignore_index : If True, do not use the index labels. verify_integrity : If True, raise Exception on creating index with duplicates

```
sr1 = pd.Series([1,-2,3])
sr2 = pd.Series([1,2,3])

sr3 = pd.concat([sr2, sr1])

print(sr3)
```

```
0    1
1    2
2    3
0    1
1   -2
2    3
dtype: int64
```

To make the index accurate:

```
sr3.reset_index(drop = True)
```

```
0    1
1    2
2    3
3    1
4   -2
5    3
dtype: int64
```

e) Astype function

Pandas astype() is the one of the most important methods. It is used to change data type of a series. When data frame is made from a csv file, the columns are imported and data type is set automatically which many times is not what it actually should have.

```
sr1

0    1
1   -2
2    3
dtype: int64
```

- You can see below int64 is mentioned

```
type(sr1[0])

numpy.int64
```

- Now you can see it is written as object

```
sr1.astype('float')

0    1.0
1   -2.0
2    3.0
dtype: float64
```

f) Between Function

Pandas between() method is used on series to check which values lie between first and second argument.

```
sr1 = pd.Series([1,2,30,4,5,6,7,8,9,20])
sr1

0     1
1     2
2    30
3     4
4     5
5     6
```

```
6     7
7     8
8     9
9    20
dtype: int64
```

```
sr1.between(10,50)
```

```
0    False
1    False
2     True
3    False
4    False
5    False
6    False
7    False
8    False
9     True
dtype: bool
```

```
g) All strings functions can be used to extract or modify texts in a
series
```

Upper and Lower Function   Len function   Strip Function   Split Function   Contains Function
Replace Function   Count Function   Startswith and Endswith Function   Find Finction

```
ser = pd.Series(["Eshant Das" , "Data Science" , "Geeks for Geeks" ,
'Hello World' , 'Machine Learning'])
```

Upper and Lower Function

```
print(ser.str.upper())
print('-'*30)
print(ser.str.lower())
```

```
0          ESHANT DAS
1        DATA SCIENCE
2     GEEKS FOR GEEKS
3         HELLO WORLD
4    MACHINE LEARNING
dtype: object
------------------------------
0          eshant das
1        data science
2     geeks for geeks
3         hello world
4    machine learning
dtype: object
```

Length function

```
for i in ser:
    print(len(i))

10
12
15
11
16
```

Strip Function

```
ser = pd.Series(["  Eshant Das" , "Data Science" , "Geeks for Geeks" ,
'Hello World' , 'Machine Learning  '])

for i in ser:
    print(i, "lenght is", len(i))

  Eshant Das lenght is 12
Data Science lenght is 12
Geeks for Geeks lenght is 15
Hello World lenght is 11
Machine Learning   lenght is 18
```

2 extra spaces has been removed

```
ser=ser.str.strip()
for i in ser:
    print(i,"Length is ",len(i))

Eshant Das Length is  10
Data Science Length is  12
Geeks for Geeks Length is  15
Hello World Length is  11
Machine Learning Length is  16
```

Split Function

```
ser

0          Eshant Das
1        Data Science
2     Geeks for Geeks
3         Hello World
4    Machine Learning
dtype: object

ser.str.split()
```

```
0          [Eshant, Das]
1        [Data, Science]
2    [Geeks, for, Geeks]
3         [Hello, World]
4    [Machine, Learning]
dtype: object
```

- IF we want to split onlt the first world of every string in the pandas series

```
ser.str.split()[0]

['Eshant', 'Das']
```

- For second word

```
ser.str.split()[1]

['Data', 'Science']
```

Contains Function

```
ser = pd.Series(["Eshant Das","Data@Science","Geeks for 
Geeks",'Hello@World','Machine Learning'])

ser.str.contains('@')

0    False
1     True
2    False
3     True
4    False
dtype: bool

for i in ser:
    for j in i:
        if j=='@':
            print(i)

Data@Science
Hello@World
```

Replace Function

```
ser.str.replace('@',' ')

0         Eshant Das
1       Data Science
2    Geeks for Geeks
3        Hello World
```

```
4     Machine Learning
dtype: object
```

Count Function

```
ser.str.count('a')

0     2
1     2
2     0
3     0
4     2
dtype: int64
```

startswith and endswith

```
ser.str.startswith('D')

0     False
1      True
2     False
3     False
4     False
dtype: bool

ser.str.endswith('s')

0      True
1     False
2      True
3     False
4     False
dtype: bool

ser.str.find('Geeks' )

0   -1
1   -1
2    0
3   -1
4   -1
dtype: int64
```

Find Function

```
ser[ser.str.find('Geeks' )!= -1]

2     Geeks for Geeks
dtype: object
```

h) Converting a Series to List

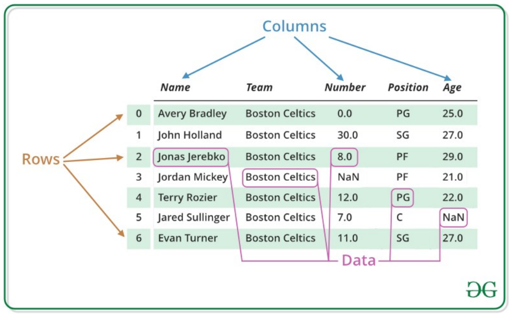
Pandas tolist() is used to convert a series to list. Initially the series is of type pandas.core.series.

```
ser.to_list()

['Eshant Das',
 'Data@Science',
 'Geeks for Geeks',
 'Hello@World',
 'Machine Learning']
```

2. Detailed Coding Implementations on Pandas DataFrame

Pandas DataFrame is two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns). A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns. Pandas DataFrame consists of three principal components, the data, rows, and columns.



	Name	Team	Number	Position	Age
0	Avery Bradley	Boston Celtics	0.0	PG	25.0
1	John Holland	Boston Celtics	30.0	SG	27.0
2	Jonas Jerebko	Boston Celtics	8.0	PF	29.0
3	Jordan Mickey	Boston Celtics	NaN	PF	21.0
4	Terry Rozier	Boston Celtics	12.0	PG	22.0
5	Jared Sullinger	Boston Celtics	7.0	C	NaN
6	Evan Turner	Boston Celtics	11.0	SG	27.0

a) Creating Data Frames

In the real world, a Pandas DataFrame will be created by loading the datasets from existing storage, storage can be SQL Database, CSV file, and Excel file. Pandas DataFrame can be created from the lists, dictionary, and from a list of dictionary etc. Dataframe can be created in different ways here are some ways by which we create a dataframe:

Creating a dataframe using List:

DataFrame can be created using a single list or a list of lists.

```
lst = ['Geeks', 'For', 'Geeks', 'is', 'portal', 'for', 'Geeks']

pd.DataFrame(lst)

        0
0   Geeks
1     For
2   Geeks
3      is
4  portal
```

```
5     for
6   Geeks
```

```
lst = [['tom',10],['jerry',12],['spike',14]]

pd.DataFrame(lst)
```

```
       0   1
0    tom  10
1  jerry  12
2  spike  14
```

Creating DataFrame from dict of ndarray/lists:

To create DataFrame from dict of narray/list, all the narray must be of same length. If index is passed then the length index should be equal to the length of arrays. If no index is passed, then by default, index will be range(n) where n is the array length.

```
data = {'name':['Tom', 'nick', 'krish', 'jack'], 'age':[20, 21, 19,
18]}


pd.DataFrame(data)
```

```
    name  age
0    Tom   20
1   nick   21
2  krish   19
3   jack   18
```

A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns. We can perform basic operations on rows/columns like selecting, deleting, adding, and renaming.

Column Selection: In Order to select a column in Pandas DataFrame, we can either access the columns by calling them by their columns name.

```
data = { 'Name'         :['Jai', 'Princi', 'Gaurav', 'Anuj'],
         'Age'          :[27, 24, 22, 32],
         'Address'      :['Delhi', 'Kanpur', 'Allahabad', 'Kannauj'],
         'Qualification':['Msc', 'MA', 'MCA', 'Phd']}

df = pd.DataFrame(data)

df[['Name', 'Qualification']]
```

```
     Name Qualification
0     Jai           Msc
1  Princi            MA
```

```
2   Gaurav           MCA
3     Anuj           Phd
```

b) Slicing in DataFrames Using iloc and loc

Pandas comprises many methods for its proper functioning. loc() and iloc() are one of those methods. These are used in slicing data from the Pandas DataFrame. They help in the convenient selection of data from the DataFrame in Python. They are used in filtering the data according to some conditions.

```
data = {'one'   : pd.Series([1, 2, 3, 4]),
        'two'   : pd.Series([10, 20, 30, 40]),
        'three' : pd.Series([100, 200, 300, 400]),
        'four'  : pd.Series([1000, 2000, 3000, 4000])}

df = pd.DataFrame(data)
df

   one  two  three  four
0    1   10    100  1000
1    2   20    200  2000
2    3   30    300  3000
3    4   40    400  4000
```

Basic loc Operations

Python loc() function The loc() function is label based data selecting method which means that we have to pass the name of the row or column which we want to select. This method includes the last element of the range passed in it, unlike iloc(). loc() can accept the boolean data unlike iloc(). Many operations can be performed using the loc() method like

```
df.loc[1:2, 'two' : 'three']

   two  three
1   20    200
2   30    300
```

Basic iloc Operations

The iloc() function is an indexed-based selecting method which means that we have to pass an integer index in the method to select a specific row/column. This method does not include the last element of the range passed in it unlike loc(). iloc() does not accept the boolean data unlike loc().

```
df.iloc[1 : -1, 1:-1 ]

   two  three
1   20    200
2   30    300
```

- you can see index 3 of both row and column has not been added here so 1 was inclusize but 3 is exclusive in the case of ilocs

Let's see another example

```
df.iloc[:,2:3]

   three
0    100
1    200
2    300
3    400
```

`Selecting Spefic Rows`

```
df.iloc[[0,2],[1,3]]

   two  four
0   10  1000
2   30  3000

df

   one  two  three  four
0    1   10    100  1000
1    2   20    200  2000
2    3   30    300  3000
3    4   40    400  4000
```

`c) Slicing Using Conditions`

Using Conditions works with loc basically

```
df.loc[df['two'] > 20, ['three','four']]

   three  four
2    300  3000
3    400  4000
```

- So we could extract only those data for which the value is more than 20
- For the columns we have used comma(,) to extract specifc columns which is 'three' and 'four'

Let's see another example

```
df.loc[df['three'] < 300, ['one','four']]

   one  four
0    1  1000
1    2  2000
```

- So you can get the inference in the same way for this code as we got for the previous code

c) Column Addition in DataFrame

```
df

   one  two  three  four
0    1   10    100  1000
1    2   20    200  2000
2    3   30    300  3000
3    4   40    400  4000
```

We can add a column in many ways. Let us discuss three ways how we can add column here

- Using List
- Using Pandas Series
- Using an existing Column(we can modify that column in the way we want and that modified part can also be displayed)

```
l = [22,33,44,55]
df['five'] = l
df

   one  two  three  four  five
0    1   10    100  1000    22
1    2   20    200  2000    33
2    3   30    300  3000    44
3    4   40    400  4000    55

sr = pd.Series([111,222,333,444])
df['six'] = sr
df

   one  two  three  four  five  six
0    1   10    100  1000    22  111
1    2   20    200  2000    33  222
2    3   30    300  3000    44  333
3    4   40    400  4000    55  444
```

Using an existing Column

```
df['seven'] = df['one'] + 10
df

   one  two  three  four  five  six  seven
0    1   10    100  1000    22  111     11
1    2   20    200  2000    33  222     12
2    3   30    300  3000    44  333     13
3    4   40    400  4000    55  444     14
```

- Now we can see the column 7 is having all the values of column 1 increamented by 10

d) Column Deletion in Dataframes

```
df

   one  two  three  four  five  six  seven
0    1   10    100  1000    22  111     11
1    2   20    200  2000    33  222     12
2    3   30    300  3000    44  333     13
3    4   40    400  4000    55  444     14
```

Using del

- You can see that the column which had the name 'six' has been deleted

```
del df['six']

df

   one  two  three  four  five  seven
0    1   10    100  1000    22     11
1    2   20    200  2000    33     12
2    3   30    300  3000    44     13
3    4   40    400  4000    55     14
```

Using pop

- You can see that the columm five has also been deleted from our dataframe

```
df.pop('five')

df

   one  two  three  four  seven
0    1   10    100  1000     11
1    2   20    200  2000     12
2    3   30    300  3000     13
3    4   40    400  4000     14

df

   one  two  three  four  seven
0    1   10    100  1000     11
1    2   20    200  2000     12
2    3   30    300  3000     13
3    4   40    400  4000     14
```

e) Addition of rows

In a Pandas DataFrame, you can add rows by using the append method. You can also create a new DataFrame with the desired row values and use the append to add the new row to the original dataframe. Here's an example of adding a single row to a dataframe:

```
df1 = pd.DataFrame([[1, 2], [3, 4]], columns = ['a','b'])
df2 = pd.DataFrame([[5, 6], [7, 8]], columns = ['a','b'])


df3 = pd.concat([df1,df2])

df3
```

```
   a  b
0  1  2
1  3  4
0  5  6
1  7  8
```

`f) Pandas drop function`

Python is a great language for doing data analysis, primarily because of the fantastic ecosystem of data-centric Python packages. Pandas is one of those packages and makes importing and analyzing data much easier.

Pandas provide data analysts a way to delete and filter data frame using .drop() method. Rows or columns can be removed using index label or column name using this method.

Syntax: DataFrame.drop(labels=None, axis=0, index=None, columns=None, level=None, inplace=False, errors='raise')

Parameters:

labels: String or list of strings referring row or column name. axis: int or string value, 0 'index' for Rows and 1 'columns' for Columns. index or columns: Single label or list. index or columns are an alternative to axis and cannot be used together. level: Used to specify level in case data frame is having multiple level index. inplace: Makes changes in original Data Frame if True. errors: Ignores error if any value from the list doesn't exists and drops rest of the values when errors = 'ignore'

Return type: Dataframe with dropped values

```
data = { 'one'   : pd.Series([1, 2, 3, 4]),
         'two'   : pd.Series([10, 20, 30, 40]),
         'three' : pd.Series([100, 200, 300, 400]),
         'four'  : pd.Series([1000, 2000, 3000, 4000])}


df = pd.DataFrame(data)
df
```

```
   one  two  three  four
0    1   10    100  1000
1    2   20    200  2000
2    3   30    300  3000
3    4   40    400  4000
```

- axis =0 => Rows (row wise)

```
df.drop([0,1], axis = 0, inplace = True)
df

   one  two  three  four
2    3   30    300  3000
3    4   40    400  4000
```

- axis =1 => Columns (column wise)

```
df.drop(['one','three'], axis = 1, inplace = True)
df

   two  four
2   30  3000
3   40  4000
```

g) Transposing a DataFrame

The .T attribute in a Pandas DataFrame is used to transpose the dataframe, i.e., to flip the rows and columns. The result of transposing a dataframe is a new dataframe with the original rows as columns and the original columns as rows.

Here's an example to illustrate the use of the .T attribute:

```
data = { 'one'   : pd.Series([1, 2, 3, 4]),
         'two'   : pd.Series([10, 20, 30, 40]),
         'three' : pd.Series([100, 200, 300, 400]),
         'four'  : pd.Series([1000, 2000, 3000, 4000])}

df = pd.DataFrame(data)
df

   one  two  three  four
0    1   10    100  1000
1    2   20    200  2000
2    3   30    300  3000
3    4   40    400  4000

df.T

          0     1     2     3
one       1     2     3     4
two      10    20    30    40
three   100   200   300   400
four   1000  2000  3000  4000
```

h) A set of more DataFrame Functionalities

```
df
```

```
   one  two  three  four
0    1   10    100  1000
1    2   20    200  2000
2    3   30    300  3000
3    4   40    400  4000
```

1. axes function

The .axes attribute in a Pandas DataFrame returns a list with the row and column labels of the DataFrame. The first element of the list is the row labels (index), and the second element is the column labels.

```
df.axes

[RangeIndex(start=0, stop=4, step=1),
 Index(['one', 'two', 'three', 'four'], dtype='object')]
```

1. ndim function

The .ndim attribute in a Pandas DataFrame returns the number of dimensions of the dataframe, which is always 2 for a DataFrame (row-and-column format).

```
df.ndim

2
```

1. dtypes

The .dtypes attribute in a Pandas DataFrame returns the data types of the columns in the DataFrame. The result is a Series with the column names as index and the data types of the columns as values.

```
df.dtypes

one      int64
two      int64
three    int64
four     int64
dtype: object
```

1. shape function

The .shape attribute in a Pandas DataFrame returns the dimensions (number of rows, number of columns) of the DataFrame as a tuple.

```
df.shape

(4, 4)
```

- 4 rows

- 4 columns

1. head() function

```
d = {
'Name'   :pd.Series(['Tom','Jerry','Spike','Popeye','Olive','Bluto','Mi
ckey']),
      'Age'    :pd.Series([10,12,14,30,28,33,15]),
      'Height':pd.Series([3.25,1.11,4.12,5.47,6.15,6.67,2.61])}

df = pd.DataFrame(d)
df

     Name  Age  Height
0     Tom   10    3.25
1   Jerry   12    1.11
2   Spike   14    4.12
3  Popeye   30    5.47
4   Olive   28    6.15
5   Bluto   33    6.67
6  Mickey   15    2.61
```

The .head() method in a Pandas DataFrame returns the first n rows (by default, n=5) of the DataFrame. This method is useful for quickly examining the first few rows of a large DataFrame to get a sense of its structure and content.

```
df.head(3)

    Name  Age  Height
0    Tom   10    3.25
1  Jerry   12    1.11
2  Spike   14    4.12
```

- By default it will display first 5 rows
- We can mention the number of starting rows we want to see
- We will see this function more often furthur since the dataframe is so small at this point so we cannot use something like df.head(20)

1. df.tail() function

The .tail() method in a Pandas DataFrame returns the last n rows (by default, n=5) of the DataFrame. This method is useful for quickly examining the last few rows of a large DataFrame to get a sense of its structure and content.

```
df.tail(3)

     Name  Age  Height
4   Olive   28    6.15
5   Bluto   33    6.67
6  Mickey   15    2.61
```

1. empty function()

The .empty attribute in a Pandas DataFrame returns a Boolean value indicating whether the DataFrame is empty or not. A DataFrame is considered empty if it has no rows.

```
df.empty
```

```
False
```

Taking Empty Data Frames

```
df1=pd.DataFrame()#Here Frame is Empty
df1.empty
```

```
True
```

`i) Statistical or Mathematical Functions`

Sum Mean Median Mode Variance Min Max Standard Deviation

```
data = {'one'   : pd.Series([1, 2, 3, 4]),
        'two'   : pd.Series([10, 20, 30, 40]),
        'three' : pd.Series([100, 200, 300, 400]),
        'four'  : pd.Series([1000, 2000, 3000, 4000])}

df = pd.DataFrame(data)
df
```

```
   one  two  three  four
0    1   10    100  1000
1    2   20    200  2000
2    3   30    300  3000
3    4   40    400  4000
```

1. Sum

```
df.sum()
```

```
one         10
two        100
three     1000
four     10000
dtype: int64
```

1. Mean

```
df.mean()
```

```
one         2.5
two        25.0
three     250.0
four     2500.0
dtype: float64
```

1. Median

```
df.median()

one         2.5
two        25.0
three     250.0
four     2500.0
dtype: float64
```

1. Mode

```
de = pd.DataFrame({'A': [1, 2, 2, 3, 4, 4, 4, 5], 'B': [10, 20, 20, 
30, 40, 40, 50, 60]})

print('A' , de['A'].mode())
print('B' , de['B'].mode())

A 0    4
Name: A, dtype: int64
B 0    20
1    40
Name: B, dtype: int64

de['A'].mode()
de['B'].mode()

0    20
1    40
Name: B, dtype: int64
```

1. Variance

```
df.var()

one      1.666667e+00
two      1.666667e+02
three    1.666667e+04
four     1.666667e+06
dtype: float64
```

1. Min

```
df.min(axis=0)

one         1
two        10
three     100
four     1000
dtype: int64

df
```

```
   one  two  three  four
0    1   10    100  1000
1    2   20    200  2000
2    3   30    300  3000
3    4   40    400  4000
```

1. Max

```
df.max()

one         4
two        40
three     400
four     4000
dtype: int64
```

1. Standard Deviation

```
df.std()

one         1.290994
two        12.909944
three     129.099445
four     1290.994449
dtype: float64
```

j) Describe Function

The describe() method in a Pandas DataFrame returns descriptive statistics of the data in the DataFrame. It provides a quick summary of the central tendency, dispersion, and shape of the distribution of a set of numerical data.

The default behavior of describe() is to compute descriptive statistics for all numerical columns in the DataFrame. If you want to compute descriptive statistics for a specific column, you can pass the name of the column as an argument.

```
data = {'one'  : pd.Series([1, 2, 3, 4]),
        'two'  : pd.Series([10, 20, 30, 40]),
        'three': pd.Series([100, 200, 300, 400]),
        'four' : pd.Series([1000, 2000, 3000, 4000]),
        'five' : pd.Series(['A','B','C','D'])}


df = pd.DataFrame(data)

df.describe()

            one        two        three         four
count  4.000000   4.000000     4.000000     4.000000
mean   2.500000  25.000000   250.000000  2500.000000
std    1.290994  12.909944   129.099445  1290.994449
```

```
min    1.000000  10.000000  100.000000  1000.000000
25%    1.750000  17.500000  175.000000  1750.000000
50%    2.500000  25.000000  250.000000  2500.000000
75%    3.250000  32.500000  325.000000  3250.000000
max    4.000000  40.000000  400.000000  4000.000000
```

`k) Pipe Functions`

1. Pipe Function

The pipe() method in a Pandas DataFrame allows you to apply a function to the DataFrame, similar to the way the apply() method works. The difference is that pipe() allows you to chain multiple operations together by passing the output of one function to the input of the next function.

```
data = {'one'  : pd.Series([1, 2, 3, 4]),
        'two'  : pd.Series([10, 20, 30, 40]),
        'three': pd.Series([100, 200, 300, 400]),
        'four' : pd.Series([1000, 2000, 3000, 4000])}

df = pd.DataFrame(data)
df

   one  two  three  four
0    1   10    100  1000
1    2   20    200  2000
2    3   30    300  3000
3    4   40    400  4000

df.mean().agg(lambda x: x**2)

C:\Users\afroz\AppData\Local\Temp\ipykernel_2244\1364954841.py:1:
FutureWarning: using <function <lambda> at 0x000001D165E48FE0> in
Series.agg cannot aggregate and has been deprecated. Use
Series.transform to keep behavior unchanged.
  df.mean().agg(lambda x: x**2)

one            6.25
two          625.00
three      62500.00
four     6250000.00
dtype: float64
```

`Example 1`

```
def add_(i,j):
    return i + j
df.pipe(add_, 10)

   one  two  three  four
0   11   20    110  1010
```

```
1   12   30    210  2010
2   13   40    310  3010
3   14   50    410  4010
```

`Example 2`

```
def mean_(col):
    return col.mean()

def square(i):
    return i ** 2

df.pipe(mean_).pipe(square)

one              6.25
two            625.00
three        62500.00
four       6250000.00
dtype: float64
```

2. Apply Function

The apply() method in a Pandas DataFrame allows you to apply a function to the DataFrame, either to individual elements or to the entire DataFrame. The function can be either a built-in Python function or a user-defined function.

```
data = {'one'  : pd.Series([1, 2, 3, 4]),
        'two'  : pd.Series([10, 20, 30, 40]),
        'three': pd.Series([100, 200, 300, 400]),
        'four' : pd.Series([1000, 2000, 3000, 4000])}

df = pd.DataFrame(data)
df

print(df.apply(np.mean))

one         2.5
two        25.0
three     250.0
four     2500.0
dtype: float64

df.mean()#Easy Way

one         2.5
two        25.0
three     250.0
four     2500.0
dtype: float64
```

```
df.apply(lambda x: x.max() - x.min())
```

```
one         3
two        30
three     300
four     3000
dtype: int64
```

```
df.max()-df.min()#Easy way
```

```
one         3
two        30
three     300
four     3000
dtype: int64
```

3. Apply map function

The map() method in a Pandas DataFrame allows you to apply a function to each element of a specific column of the DataFrame. The function can be either a built-in Python function or a user-defined function.

```
df.applymap(lambda x : x*100)
```

```
C:\Users\afroz\AppData\Local\Temp\ipykernel_2244\795461004.py:1:
FutureWarning: DataFrame.applymap has been deprecated. Use
DataFrame.map instead.
  df.applymap(lambda x : x*100)
```

```
   one   two  three    four
0  100  1000  10000  100000
1  200  2000  20000  200000
2  300  3000  30000  300000
3  400  4000  40000  400000
```

```
applymap and apply are both functions in the pandas library used for
applying a function to elements of a pandas DataFrame or Series.

applymap is used to apply a function to every element of a DataFrame.
It returns a new DataFrame where each element has been modified by the
input function.

apply is used to apply a function along any axis of a DataFrame or
Series. It returns either a Series or a DataFrame, depending on the
axis along which the function is applied and the return value of the
function. Unlike applymap, apply can take into account the context of
the data, such as the row or column label.

So, applymap is meant for element-wise operations while apply can be
used for both element-wise and row/column-wise operations.
```

```
df = pd.DataFrame({ 'A': [1.2, 3.4, 5.6],
                    'B': [7.8, 9.1, 2.3]})

df_1 = df.applymap(np.int64)
print(df_1)

df_2 = df.apply(lambda row : row.mean(), axis = 0)
print(df_2)
```

```
   A  B
0  1  7
1  3  9
2  5  2
A    3.4
B    6.4
dtype: float64
```

```
C:\Users\afroz\AppData\Local\Temp\ipykernel_2244\2749802798.py:4:
FutureWarning: DataFrame.applymap has been deprecated. Use
DataFrame.map instead.
  df_1 = df.applymap(np.int64)
```

`l) Reindex Function`

The reindex function in Pandas is used to change the row labels and/or column labels of a DataFrame. This function can be used to align data from multiple DataFrames or to update the labels based on new data. The function takes in a list or an array of new labels as its first argument and, optionally, a fill value to replace any missing values. The reindexing can be done along either the row axis (0) or the column axis (1). The reindexed DataFrame is returned.

`Example 1 - Rows`

```
data = { 'one'   : pd.Series([1, 2, 3, 4]),
         'two'   : pd.Series([10, 20, 30, 40]),
         'three' : pd.Series([100, 200, 300, 400]),
         'four'  : pd.Series([1000, 2000, 3000, 4000])}

df = pd.DataFrame(data)

print(df)
print('-'*30)
print(df.reindex([1,0,3,2]))
```

```
   one  two  three  four
0    1   10    100  1000
1    2   20    200  2000
2    3   30    300  3000
3    4   40    400  4000
------------------------------
   one  two  three  four
```

```
1    2   20    200  2000
0    1   10    100  1000
3    4   40    400  4000
2    3   30    300  3000
```

Example 2 - Columns

```
data = {'Name' : ['John', 'Jane', 'Jim', 'Joan'],
        'Age'  : [25, 30, 35, 40],
        'City' : ['New York', 'Los Angeles', 'Chicago', 'Houston']}

df = pd.DataFrame(data)

df.reindex(columns = ['Name','City','Age'])

   Name         City  Age
0  John     New York   25
1  Jane  Los Angeles   30
2   Jim      Chicago   35
3  Joan      Houston   40
```

m) Renaming Columns in Pandas DataFrame

The rename function in Pandas is used to change the row labels and/or column labels of a DataFrame. It can be used to update the names of one or multiple rows or columns by passing a dictionary of new names as its argument. The dictionary should have the old names as keys and the new names as values

```
data = { 'one'   : pd.Series([1, 2, 3, 4]),
         'two'   : pd.Series([10, 20, 30, 40]),
         'three' : pd.Series([100, 200, 300, 400]),
         'four'  : pd.Series([1000, 2000, 3000, 4000])}

df = pd.DataFrame(data)

df.rename(columns = {'one' : 'One','two': 'Two', 'three' : 'Three',
'four' : 'Four'},
           inplace = True, index = {0:'a',1:'b',2:'c',4:'d'})
df

   One  Two  Three  Four
a    1   10    100  1000
b    2   20    200  2000
c    3   30    300  3000
3    4   40    400  4000
```

n) Sorting in Pandas DataFrame

Pandas provides several methods to sort a DataFrame based on one or more columns.

- sort_values: This method sorts the DataFrame based on one or more columns. The default sorting order is ascending, but you can change it to descending by passing the ascending argument with a value of False. bash

```
data = { 'one'   : pd.Series([11, 51, 31, 41]),
         'two'   : pd.Series([10, 50, 30, 40]),
         'three' : pd.Series([100, 200, 500, 400]),
         'four'  : pd.Series([1000, 2000, 3000, 4000])}

df = pd.DataFrame(data)
df

   one  two  three  four
0   11   10    100  1000
1   51   50    200  2000
2   31   30    500  3000
3   41   40    400  4000
```

```
Sort with respect to Scecific Column
```

```
df.sort_values(by = 'two')

   one  two  three  four
0   11   10    100  1000
2   31   30    500  3000
3   41   40    400  4000
1   51   50    200  2000
```

```
Sort in Scecific Order
```

```
df.sort_values(by = 'one', ascending = False)

   one  two  three  four
1   51   50    200  2000
3   41   40    400  4000
2   31   30    500  3000
0   11   10    100  1000
```

```
Sort in Scecific Order based on multiple Columns
```

```
df.sort_values(by = ['three','one'])

   one  two  three  four
0   11   10    100  1000
1   51   50    200  2000
3   41   40    400  4000
2   31   30    500  3000
```

```
Sort with Specific Sorting Algorithm:<br>
```

- quicksort

- mergesort
- heapsort

```
df.sort_values(by=['one'], kind='heapsort')  # □ Correct spelling

   one  two  three  four
0   11   10    100  1000
2   31   30    500  3000
3   41   40    400  4000
1   51   50    200  2000
```

o) Groupby Functions

The groupby function in pandas is used to split a dataframe into groups based on one or more columns. It returns a DataFrameGroupBy object, which is similar to a DataFrame but has some additional methods to perform operations on the grouped data.

```
cricket = {'Team'   : ['India', 'India', 'Australia', 'Australia',
'SA', 'SA', 'SA', 'SA', 'NZ', 'NZ', 'NZ', 'India'],
           'Rank'   : [2, 3, 1,2, 3,4 ,1 ,1,2 , 4,1,2],
           'Year'   :
[2014,2015,2014,2015,2014,2015,2016,2017,2016,2014,2015,2017],
           'Points' :
[876,801,891,815,776,784,834,824,758,691,883,782]}

df = pd.DataFrame(cricket)
df

         Team  Rank  Year  Points
0       India     2  2014     876
1       India     3  2015     801
2   Australia     1  2014     891
3   Australia     2  2015     815
4          SA     3  2014     776
5          SA     4  2015     784
6          SA     1  2016     834
7          SA     1  2017     824
8          NZ     2  2016     758
9          NZ     4  2014     691
10         NZ     1  2015     883
11      India     2  2017     782

df.groupby('Team').groups

{'Australia': [2, 3], 'India': [0, 1, 11], 'NZ': [8, 9, 10], 'SA': [4,
5, 6, 7]}
```

- Austrealia is present in index 2 and 3
- India is present in index 0,1 and 11 and so on

`To search for specific Country with specific year`

```
df.groupby(['Team','Year']).get_group(('Australia',2014))
```

```
        Team  Rank  Year  Points
2  Australia     1  2014     891
```

```
df[(df['Year']==2014) & (df['Team']=='Australia')] #Easy and Underatsand Code
```

```
        Team  Rank  Year  Points
2  Australia     1  2014     891
```

If the data is not present then we will be getting an error

Adding some statistical computation on top of groupby

```
df.groupby('Team').sum()['Points']
```

```
Team
Australia    1706
India        2459
NZ           2332
SA           3218
Name: Points, dtype: int64
```

```
df.groupby('Team').agg({'Points':'sum'}).sort_values(by='Points',ascending = False)
```

```
           Points
Team
SA           3218
India        2459
NZ           2332
Australia    1706
```

- This means we have displayed the teams which are having the maximum sum in Poitns

Let us sort it to get it in a better way

```
df.groupby('Team').sum()['Points'].sort_values(ascending = False)
```

```
Team
SA           3218
India        2459
NZ           2332
Australia    1706
Name: Points, dtype: int64
```

Checking multiple stats for points team wise

```
groups = df.groupby('Team')
groups['Points'].agg([np.sum, np.mean, np.std,np.max,np.min])
```

```
C:\Users\afroz\AppData\Local\Temp\ipykernel_2244\2356013916.py:2:
FutureWarning: The provided callable <function sum at
0x000001D164EF0FE0> is currently using SeriesGroupBy.sum. In a future
version of pandas, the provided callable will be used directly. To
keep current behavior pass the string "sum" instead.
  groups['Points'].agg([np.sum, np.mean, np.std,np.max,np.min])
C:\Users\afroz\AppData\Local\Temp\ipykernel_2244\2356013916.py:2:
FutureWarning: The provided callable <function mean at
0x000001D164EF23E0> is currently using SeriesGroupBy.mean. In a future
version of pandas, the provided callable will be used directly. To
keep current behavior pass the string "mean" instead.
  groups['Points'].agg([np.sum, np.mean, np.std,np.max,np.min])
C:\Users\afroz\AppData\Local\Temp\ipykernel_2244\2356013916.py:2:
FutureWarning: The provided callable <function std at
0x000001D164EF2520> is currently using SeriesGroupBy.std. In a future
version of pandas, the provided callable will be used directly. To
keep current behavior pass the string "std" instead.
  groups['Points'].agg([np.sum, np.mean, np.std,np.max,np.min])
C:\Users\afroz\AppData\Local\Temp\ipykernel_2244\2356013916.py:2:
FutureWarning: The provided callable <function max at
0x000001D164EF19E0> is currently using SeriesGroupBy.max. In a future
version of pandas, the provided callable will be used directly. To
keep current behavior pass the string "max" instead.
  groups['Points'].agg([np.sum, np.mean, np.std,np.max,np.min])
C:\Users\afroz\AppData\Local\Temp\ipykernel_2244\2356013916.py:2:
FutureWarning: The provided callable <function min at
0x000001D164EF1B20> is currently using SeriesGroupBy.min. In a future
version of pandas, the provided callable will be used directly. To
keep current behavior pass the string "min" instead.
  groups['Points'].agg([np.sum, np.mean, np.std,np.max,np.min])
```

```
            sum        mean        std  max  min
Team
Australia  1706  853.000000  53.740115  891  815
India      2459  819.666667  49.702448  876  782
NZ         2332  777.333333  97.449132  883  691
SA         3218  804.500000  28.769196  834  776
```

filter function along with groupby

```
df.groupby('Team').filter(lambda x : len(x) == 4)
```

```
  Team  Rank  Year  Points
4   SA     3  2014     776
5   SA     4  2015     784
6   SA     1  2016     834
7   SA     1  2017     824
```

- The data of South Africa are present equal to 4 times that is why South Africa is being displayed here

```
df.groupby('Team').filter(lambda x : len(x) == 3)

     Team  Rank  Year  Points
0   India     2  2014     876
1   India     3  2015     801
8      NZ     2  2016     758
9      NZ     4  2014     691
10     NZ     1  2015     883
11  India     2  2017     782
```

- The data of India and New Zealand are present 3 times so that is why they are being displayed here

```
df.groupby('Team').filter(lambda x : len(x) == 2 or len(x)==3)

         Team  Rank  Year  Points
0       India     2  2014     876
1       India     3  2015     801
2   Australia     1  2014     891
3   Australia     2  2015     815
8          NZ     2  2016     758
9          NZ     4  2014     691
10         NZ     1  2015     883
11      India     2  2017     782
```

3. Working with csv files and basic data Analysis Using Pandas

```
a) Reading csv
```

Reading csv files from local system

```
df = pd.read_csv('Football.csv')

df.head()

  Country   League   Club       Player Names  Matches_Played
Substitution   \
0   Spain  La Liga  (BET)    Juanmi Callejon              19
16
1   Spain  La Liga  (BAR)  Antoine Griezmann              36
0
2   Spain  La Liga  (ATL)        Luis Suarez              34
1
3   Spain  La Liga  (CAR)       Ruben Castro              32
3
4   Spain  La Liga  (VAL)      Kevin Gameiro              21
```

```
10

   Mins  Goals     xG  xG Per Avg Match  Shots  OnTarget  Shots Per
Avg Match  \
0  1849     11   6.62              0.34     48        20
2.47
1  3129     16  11.86              0.36     88        41
2.67
2  2940     28  23.21              0.75    120        57
3.88
3  2842     13  14.06              0.47    117        42
3.91
4  1745     13  10.65              0.58     50        23
2.72

   On Target Per Avg Match  Year
0                     1.03  2016
1                     1.24  2016
2                     1.84  2016
3                     1.40  2016
4                     1.25  2016
```

Reading CSV files from github repositories **NOTE:** The link of the page should be copied when the file is in raw format

```
link = 'https://raw.githubusercontent.com/AshishJangra27/Data-
Analysis-with-Python-GFG/main/3.%20Data%20Preprocessing%20-%20Removing
%20Null%20Value%20Rows/googleplaystore.csv'

# df = pd.read_csv(link)
# df.head()

#
test='https://raw.githubusercontent.com/prasertcbs/basic-dataset/refs/
heads/master/Employee%20data.csv'
# te=pd.read_csv(test)
# te.head()
```

`b) Pandas Info Function`

Pandas dataframe.info() function is used to get a concise summary of the dataframe. It comes really handy when doing exploratory analysis of the data. To get a quick overview of the dataset we use the dataframe.info() function.

Syntax: DataFrame.info(verbose=None, buf=None, max_cols=None, memory_usage=None, null_counts=None)

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 660 entries, 0 to 659
Data columns (total 15 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   Country                  660 non-null    object 
 1   League                   660 non-null    object 
 2   Club                     626 non-null    object 
 3   Player Names             660 non-null    object 
 4   Matches_Played           660 non-null    int64  
 5   Substitution             660 non-null    int64  
 6   Mins                     660 non-null    int64  
 7   Goals                    660 non-null    int64  
 8   xG                       660 non-null    float64
 9   xG Per Avg Match         660 non-null    float64
 10  Shots                    660 non-null    int64  
 11  OnTarget                 660 non-null    int64  
 12  Shots Per Avg Match      660 non-null    float64
 13  On Target Per Avg Match  660 non-null    float64
 14  Year                     660 non-null    int64  
dtypes: float64(4), int64(7), object(4)
memory usage: 77.5+ KB
```

c) isnull() function to check if there are nan values present

```
df.isnull()

     Country  League   Club  Player Names  Matches_Played
Substitution   \
0      False   False  False         False           False
False
1      False   False  False         False           False
False
2      False   False  False         False           False
False
3      False   False  False         False           False
False
4      False   False  False         False           False
False
..       ...     ...    ...           ...             ...            .
..
655    False   False  False         False           False
False
656    False   False  False         False           False
False
657    False   False  False         False           False
False
658    False   False   True         False           False
False
659    False   False  False         False           False
```

```
False

      Mins  Goals     xG  xG Per Avg Match  Shots  OnTarget  \
0    False  False  False             False  False     False   
1    False  False  False             False  False     False   
2    False  False  False             False  False     False   
3    False  False  False             False  False     False   
4    False  False  False             False  False     False   
..     ...    ...    ...               ...    ...       ...   
655  False  False  False             False  False     False   
656  False  False  False             False  False     False   
657  False  False  False             False  False     False   
658  False  False  False             False  False     False   
659  False  False  False             False  False     False   

     Shots Per Avg Match  On Target Per Avg Match   Year
0                  False                    False  False
1                  False                    False  False
2                  False                    False  False
3                  False                    False  False
4                  False                    False  False
..                   ...                      ...    ...
655                False                    False  False
656                False                    False  False
657                False                    False  False
658                False                    False  False
659                False                    False  False

[660 rows x 15 columns]
```

So we can see we are getting a boolean kind of a table giving True and False

If we use the `sum` function along with it then we can get how many null values are present in each columns

```
df.isnull().sum()

Country                     0
League                      0
Club                       34
Player Names                0
Matches_Played              0
Substitution                0
Mins                        0
Goals                       0
xG                          0
xG Per Avg Match            0
Shots                       0
OnTarget                    0
Shots Per Avg Match         0
```

```
On Target Per Avg Match     0
Year                        0
dtype: int64
```

d) Quantile function to get the specific percentile value

Let us check the 80 percentile value of each columns using describe function first

```
df.describe(percentiles = [.80])
```

```
       Matches_Played  Substitution          Mins       Goals
xG  \
count      660.000000     660.000000   660.000000  660.000000
660.000000
mean        22.371212       3.224242  2071.416667   11.810606
10.089606
std          9.754658       3.839498   900.595049    6.075315
5.724844
min          2.000000       0.000000   264.000000    2.000000
0.710000
50%         24.000000       2.000000  2245.500000   11.000000
9.285000
80%         32.000000       6.000000  2915.800000   15.000000
14.076000
max         38.000000      26.000000  4177.000000   42.000000
32.540000

       xG Per Avg Match       Shots    OnTarget  Shots Per Avg
Match  \
count        660.000000  660.000000  660.000000           660.000000

mean           0.476167   64.177273   28.365152             2.948015

std            0.192831   34.941622   16.363149             0.914906

min            0.070000    5.000000    2.000000             0.800000

50%            0.435000   62.000000   26.000000             2.845000

80%            0.610000   90.000000   39.000000             3.600000

max            1.350000  208.000000  102.000000             7.200000


       On Target Per Avg Match         Year
count               660.000000   660.000000
mean                  1.315652  2018.363636
std                   0.474239     1.367700
min                   0.240000  2016.000000
50%                   1.250000  2019.000000
```

```
80%                       1.630000  2020.000000
max                       3.630000  2020.000000
```

So we can see the 80th Percentile value of Mins is 2915.80

Let us use the quantile function to get the exact value now

```
df['Mins'].quantile(.80)

np.float64(2915.8)
```

Here we go we got the same value

```
To get the 99 percentile value we can write
```

```
df['Mins'].quantile(.99)

np.float64(3520.0199999999995)
```

- This funciton is important as it can be used to treat ourliers in Data Science EDA process

```
e) Copy function
```

If we normal do: de=df Then a change in `de` will affect the data of `df` as well so we need to copy in such a way that it creates a totally new object and does not affect the old dataframe

```
de = df.copy()
de.head(3)

  Country   League   Club        Player Names  Matches_Played  
Substitution   \
0   Spain  La Liga  (BET)     Juanmi Callejon              19
16
1   Spain  La Liga  (BAR)  Antoine Griezmann              36
0
2   Spain  La Liga  (ATL)         Luis Suarez              34
1

   Mins  Goals     xG  xG Per Avg Match  Shots  OnTarget  Shots Per
Avg Match  \
0  1849     11   6.62              0.34     48        20
2.47
1  3129     16  11.86              0.36     88        41
2.67
2  2940     28  23.21              0.75    120        57
3.88

   On Target Per Avg Match  Year
0                     1.03  2016
1                     1.24  2016
2                     1.84  2016
```

```
de['Year+100'] = de['Year'] + 100
de.head()

  Country   League   Club        Player Names  Matches_Played  
Substitution   \
0   Spain  La Liga  (BET)     Juanmi Callejon              19  
16
1   Spain  La Liga  (BAR)   Antoine Griezmann              36  
0
2   Spain  La Liga  (ATL)         Luis Suarez              34  
1
3   Spain  La Liga  (CAR)        Ruben Castro              32  
3
4   Spain  La Liga  (VAL)       Kevin Gameiro              21  
10

   Mins  Goals     xG  xG Per Avg Match  Shots  OnTarget  Shots Per 
Avg Match  \
0  1849     11   6.62              0.34     48        20  
2.47
1  3129     16  11.86              0.36     88        41  
2.67
2  2940     28  23.21              0.75    120        57  
3.88
3  2842     13  14.06              0.47    117        42  
3.91
4  1745     13  10.65              0.58     50        23  
2.72

   On Target Per Avg Match  Year  Year+100  
0                     1.03  2016      2116  
1                     1.24  2016      2116  
2                     1.84  2016      2116  
3                     1.40  2016      2116  
4                     1.25  2016      2116  
```

- So we can see a new column has been added here but our old data is secured

```
df.head()

  Country   League   Club        Player Names  Matches_Played  
Substitution   \
0   Spain  La Liga  (BET)     Juanmi Callejon              19  
16
1   Spain  La Liga  (BAR)   Antoine Griezmann              36  
0
2   Spain  La Liga  (ATL)         Luis Suarez              34  
1
3   Spain  La Liga  (CAR)        Ruben Castro              32  
3
```

```
4   Spain  La Liga  (VAL)      Kevin Gameiro              21
10

   Mins  Goals     xG  xG Per Avg Match  Shots  OnTarget  Shots Per
Avg Match  \
0  1849     11   6.62              0.34     48        20
2.47
1  3129     16  11.86              0.36     88        41
2.67
2  2940     28  23.21              0.75    120        57
3.88
3  2842     13  14.06              0.47    117        42
3.91
4  1745     13  10.65              0.58     50        23
2.72

   On Target Per Avg Match  Year
0                     1.03  2016
1                     1.24  2016
2                     1.84  2016
3                     1.40  2016
4                     1.25  2016
```

- The new column is not present here

```
f) Value Counts function
```

Pandas Series.value_counts() function return a Series containing counts of unique values. The resulting object will be in descending order so that the first element is the most frequently-occurring element. Excludes NA values by default.

Syntax: Series.value_counts(normalize=False, sort=True, ascending=False, bins=None, dropna=True)

```
df['Player Names'].value_counts()

Player Names
Lionel Messi          5
Luis Suarez           5
Fabio Quagliarella    5
Andrea Belotti        5
Robert Lewandowski    5
                     ..
Robson                1
Renato Kayzer         1
Donny van de Beek     1
Teun Koopmeiners      1
Cantalapiedra         1
Name: count, Length: 444, dtype: int64
```

g) Unique and Nunique Function

While analyzing the data, many times the user wants to see the unique values in a particular column, which can be done using Pandas unique() function.

```
df['Player Names'].unique()

array(['Juanmi Callejon', 'Antoine Griezmann', 'Luis Suarez',
       'Ruben Castro', 'Kevin Gameiro', 'Cristiano Ronaldo',
       'Karim Benzema', 'Neymar ', 'Iago Aspas', 'Sergi Enrich',
       'Aduriz ', 'Sandro Ramlrez', 'Lionel Messi', 'Gerard Moreno',
       'Morata', 'Wissam Ben Yedder', 'Willian Jose', 'Andone ',
       'Cedric Bakambu', 'Isco', 'Mohamed Salah', 'Gregoire Defrel',
       'Ciro Immobile', 'Nikola Kalinic', 'Dries Mertens',
       'Alejandro Gomez', 'Jose CallejOn', 'Iago Falque',
       'Giovanni Simeone', 'Mauro Icardi', 'Diego Falcinelli',
       'Cyril Thereau', 'Edin Dzeko', 'Lorenzo Insigne',
       'Fabio Quagliarella', 'Borriello ', 'Carlos Bacca',
       'Gonzalo Higuain', 'Keita Balde', 'Andrea Belotti', 'Fin
Bartels',
       'Lars Stindl', 'Serge Gnabry', 'Wagner ', 'Andrej Kramaric',
       'Florian Niederlechner', 'Robert Lewandowski', 'Emil Forsberg',
       'Timo Werner', 'Nils Petersen', 'Vedad Ibisevic', 'Mario
Gomez',
       'Maximilian Philipp', 'A\x81dam Szalai',
       'Pierre-Emerick Aubameyang', 'Guido Burgstaller', 'Max Kruse',
       'Chicharito ', 'Anthony Modeste', 'Arjen Robben', 'Alexis
Sanchez',
       'Romelu Lukaku', 'Harry Kane', 'Jamie Vardy', 'Christian
Benteke',
       'Pedro None', 'Eden Hazard', 'Roberto Firmino', 'Sadio Mane',
       'Philippe Coutinho', 'Diego Costa', 'Dele Alli', 'Sergio
Aguero',
       'Jermain Defoe', 'Fernando Llorente', 'Michail Antonio',
       'Zlatan Ibrahimovic', 'Olivier Giroud', 'Son Heung-Min',
       'Joshua King', 'Diego Souza', 'Pablo ', 'Robinho ', 'Kempes ',
       'Gabriel Jesus', 'Bruno Rangel ', 'Rogerio ', 'Vitor Bueno',
       'Marinho ', 'Grafite ', 'Andres Chavez', 'Cicero Semedo',
'Sassa',
       'Giorgian de Arrascaeta', 'Keno ', 'Fred ', 'Kleber Gladiador',
       'Pottker ', 'Jonathan Copete', 'Ricardo Oliveira',
       'Angel Rodriguez', 'Gareth Bale', 'Rodrigo None', 'Sergio
Leon',
       'Maxi Gomez', 'Mikel Oyarzabal', 'Willian Josa', 'Simone Zaza',
       'Portu ', 'Cristhian Stuani', 'Santi Mina', 'Morales ', 'Munir
',
       'Jose Callejon', 'Sergej Milinkovic-Savic', 'Duvan Zapata',
       'Paulo Dybala', 'Mirco Antenucci', 'Luis Alberto',
       'Roberto Inglese', 'Josip Ilicic', 'Ivan Perisic',
       'Leonardo Pavoletti', 'Kevin Lasagna', 'Niclas Fullkrug',
       'Salomon Kalou', 'Thorgan Hazard', 'Jean-Kevin Augustin',
       'Sandro Wagner ', 'Leon Bailey', 'Daniel Didavi',
       'Alfred Finnbogason', 'Davie Selke', 'Mark Uth',
       'Michael Gregoritsch', 'Julian Brandt', 'Kevin Volland', 'Roger
',
       'Arthur Caike', 'Everton ', 'Hernanes ', 'Luiz Fernando',
       'Wellington Paulista', 'Santiago Trellez', 'Jo', 'Thiago
Neves',
       'Bruno Henrique', 'Dudu ', 'Diego ', 'Lucca ', 'Henrique
Dourado',
       'Andre', 'Edigar Junio ', 'Junior  Dutra', 'Jaime Mata',
       'Inaki Williams', 'Chimy Avila', 'Raul de Tomas', 'Pablo
Sarabia',
       'Borja Iglesias', 'Jorge Molina', 'Charles', 'Arkadiusz Milik',
       'Mandzukic None', 'Andrea Petagna', 'Francesco Caputo',
       'Stephan El Shaarawy', 'Gervinho ', 'Krzysztof Piatek',
       'Alassane Plea', 'Kai Havertz', 'Luka Jovic', 'Ante Rebic',
       'Jadon Sancho', 'Ondrej Duda', 'Paco Alcacer', 'Benito Raman',
       'Wout Weghorst', 'Ishak Belfodil', 'Marco Reus',
       'Jean-Philippe Mateta', 'Sebastien Haller', 'Yussuf Poulsen',
       'Angel Di Maria', 'Remi Oudin', 'Nicolas Pepe', 'Emiliano
Sala',
       'Jonathan Bamba', "M'Baye Niang", 'Edinson Cavani',
       'Stephane Bahoken', 'Max Gradel', 'Florian Thauvin',
       'Kylian Mbappe-Lottin', 'Wahbi Khazri', 'Falcao ',
       'Gaetan Laborde', 'Andy Delort', 'Moussa Dembele', 'Memphis
Depay',
       'Lebo Mothiba', 'Francois Kamano', 'Luka Milivojevic',
       'Paul Pogba', 'Ashley Barnes', 'Glenn Murray', 'Richarlison ',
       'Callum Wilson', 'Gylfi Sigurdsson', 'Raheem Sterling',
       'Ayoze Perez', 'Alexandre Lacazette', 'Raul Jimenez',
       'Lucas Paqueta', 'Juan Cazares', 'Deyverson ', 'Leandro Damiao
',
       'Yago Pikachu', 'Rodrygo ', 'Andres Rios', 'Roger Guedes ',
       'Leandro Pereira ', 'Pedro ', 'Nico Lupez', 'Nene ', 'Gilberto
',
       'Henrique ', 'Willian ', 'Gabriel Barbosa', 'Diego Rossi',
       'Josef Martinez', 'Chris Wondolowski', 'Nani ', 'Kacper
Przybylko',
       'Kei Kamara', 'C.J. Sapong', 'Gyasi Zardes', 'Heber ',
       'Mauro Manotas', 'Brian Fernandez', 'Alejandro Pozuelo',
       'Felipe Gutierrez', 'Jordan Morris', 'Raul Ruidiaz',
       'Jozy Altidore', 'Carlos Vela', 'Nemanja Nikolic',
       'Alexandru Mitrita', 'Joselu', 'Carlos Fernandez', 'Ante
Budimir',
       'Lucas Perez', 'Loren Moron', 'Raul Garcla', 'Morata ',
       'Sergio Ramos', 'Lucas Ocampos', 'Cazorla ', 'Pote ',
       'Tiquinho Soares', 'Eduardo Mancha', 'Paulinho ', 'Alex
Telles',
       'Bruno Viana', 'Mehrdad Mohammadi', 'Carlos Valenzuela',
       'Ruben Lameiras', 'Moussa Marega', 'Gian-Luca Waldschmidt',
       'Samuel Lino', 'Andre Andre', 'Mehdi Taremi', 'Carlos
Vinicius',
       'Sergio Oliveira', 'Douglas Tanque', 'Fabio Abreu',
       'Brayan Riascos', 'Alex Telles ', 'Fabio Martins',
       'Haris  Seferovic', 'Joao Teixeira', 'Bruno Fernandes',
       'Angel Gomes', 'Toni MartÃ\xadnez', 'Pizzi ', 'Bozhidar Kraev',
       'Sandro Lima', 'Rodrigo Pinho', 'Thiago Santana', 'Trincao ',
       'Andraz Sporar', 'Ricardo Horta', 'Bruno Duarte', 'Nuno
Santos',
       'Domenico Berardi', 'Joao Pedro', 'Andreas Cornelius',
       'Marco Mancosu', 'Lautaro Martrinez', 'Luis Muriel',
       'Gianluca Lapadula', 'Marcus Thuram', 'Rouwen Hennings',
       'Andre Silva', 'Erling Haaland', 'Jhon Cordoba', 'Robin
Quaison',
       'Sebastian Andersson', 'Dimitri Payet', 'Kasper Dolberg',
       'Adrien Thomasson', 'Dario Benedetto', 'Ludovic Ajorque',
       'Islam Slimani', 'Adrien Hunou', 'Denis Bouanga',
       'Sehrou Guirassy', 'Ã\x81ngel Di Maria', 'Habib Diallo',
       'Victor Osimhen', 'Dominic Calvert-Lewin', 'Kevin De Bruyne',
       'Chris Wood', 'Anthony Martial', 'Riyad Mahrez', 'Marcus
Rashford',
       'Danny Ings', 'Richarlison  ', 'Teemu Pukki', 'Tammy Abraham',
       'Thiago Galhardo', 'Paolo Guerrero', 'Pepe', 'Michael ',
       'Carlos Sanchez', 'Everaldo ', 'Artur ', 'Marcelo  Cirino',
       'Yeferson Soteldo', 'Eduardo Sasha', 'Rafael Moura', 'Antony
None',
       'Quincy Promes', 'Dusan Tadic', 'Armando Broja', 'Steven
Berghuis',
       'Michael de Leeuw', 'Lois Openda', 'Danilo Nome', 'Lennart
Thy',
       'Donyell Malen', 'Noni Madueke', 'Davy Klaassen',
       'Oussama Tannane', 'Vaclav Cerny', 'Vangelis Pavlidis',
       'Henk Veerman', 'Abdou Harroui', 'Rai Vloet', 'Lassina Traore',
       'Georgios Giakoumakis', 'Alex Pozuelo', 'Kevin Molino',
       'Damir Kreilach', 'Bradley Wright-Phillips', 'Nicolas Lodeiro',
       'Daryl Dike', 'Cristian Pavon', 'Chris Mueller', 'Romell
Quioto',
       'Gustavo Bou', 'Robert Beric', 'Ayo Akinola', 'Jeremy
Ebobisse',
       'Diego Valeri', 'Youssef   En-Nesyri', 'Carlos Soler',
       'Cristian Tello', 'Esteban Burgos', 'Joao Felix',
       'Federico Valverde', 'Kike GarcIa', 'Ansu Fati', 'Roberto
Soriano',
       'Gaetano Castrovilli', 'Henrikh Mkhitaryan', 'Jordan Veretout',
       'Lautaro MartInez', 'Hirving Lozano', 'Lucas Alario', 'Bas
Dost',
       'Dani Olmo', 'Ellyes Skhiri', 'Thomas Muller', 'Andre Hahn',
       'Daniel Caligiuri', 'Matheus Cunha', 'Ludovic Blas', 'Karl
Toko',
       'Burak Yilmaz', 'Ibrahima Niane', 'Boulaye Dia', 'Moise Kean',
       'Ignatius Ganago', 'Irvin Cardona', 'Wissam Ben', 'Amine
Gouiri',
       'Mama Balde', 'Gael Kakuta', 'James Ward-Prowse', 'Diogo Jota',
       'Wilfried Zaha', 'Jack Grealish', 'Jarrod Bowen',
       'Patrick Bamford', 'Danny Ings ', 'Neal Maupay', 'Ollie
Watkins',
       'Luciano ', 'Vinicius ', 'Raphael Veiga', 'Luiz Adriano',
       'Cleber ', 'German Cano', 'Brenner None', 'Matheus Babi',
       'Alerrandro ', 'Claudinho ', 'Robson', 'Renato Kayzer',
       'Donny van de Beek', 'Teun Koopmeiners', 'Cantalapiedra ',
       'Bryan Linssen', 'Matavz ', 'Oussama Idrissi', 'Chidera Ejuke',
       'Myron Boadu', 'Klaas-Jan Huntelaar', 'Haris Vuckic',
       'Gyrano Kerk', 'Denzel Dumfries', 'Cyriel Dessers ', 'Cody
Gakpo'],
      dtype=object)
```

While analyzing the data, many times the user wants to see the unique values in a particular column. Pandas nunique() is used to get a count of unique values.

```
df['Player Names'].nunique()

444
```

`h) dropna() function`

Sometimes csv file has null values, which are later displayed as NaN in Data Frame. Pandas dropna() method allows the user to analyze and drop Rows/Columns with Null values in different ways.

Syntax:

DataFrameName.dropna(axis=0,inplace=False)

axis: axis takes int or string value for rows/columns. Input can be 0 or 1 for Integer and ‘index’ or ‘columns’ for String.

```
link = 'https://raw.githubusercontent.com/AshishJangra27/Data-
Analysis-with-Python-GFG/main/3.%20Data%20Preprocessing%20-%20Removing
%20Null%20Value%20Rows/googleplaystore.csv'

df = pd.read_csv(link)
df.head()

                                                 App        Category
Rating  \
0     Photo Editor & Candy Camera & Grid & ScrapBook   ART_AND_DESIGN
4.1
```

```
1                                 Coloring book moana  ART_AND_DESIGN
3.9
2  U Launcher Lite – FREE Live Cool Themes, Hide ...  ART_AND_DESIGN
4.7
3                               Sketch - Draw & Paint  ART_AND_DESIGN
4.5
4               Pixel Draw - Number Art Coloring Book  ART_AND_DESIGN
4.3

  Reviews  Size     Installs  Type Price Content Rating  \
0     159   19M      10,000+  Free     0       Everyone
1     967   14M     500,000+  Free     0       Everyone
2   87510  8.7M   5,000,000+  Free     0       Everyone
3  215644   25M  50,000,000+  Free     0           Teen
4     967  2.8M     100,000+  Free     0       Everyone

                      Genres      Last Updated         Current Ver  \
0               Art & Design   January 7, 2018               1.0.0
1  Art & Design;Pretend Play  January 15, 2018               2.0.0
2               Art & Design    August 1, 2018               1.2.4
3               Art & Design      June 8, 2018  Varies with device
4    Art & Design;Creativity     June 20, 2018                 1.1

    Android Ver
0  4.0.3 and up
1  4.0.3 and up
2  4.0.3 and up
3    4.2 and up
4    4.4 and up
```

```
df.isnull().sum()
```

```
App                  0
Category             0
Rating            1474
Reviews              0
Size                 0
Installs             0
Type                 1
Price                0
Content Rating       1
Genres               0
Last Updated         0
Current Ver          8
Android Ver          3
dtype: int64
```

- ok so it seems like we have alot of Null Values in column Rating and few null values in some other columns

```
df.dropna(inplace = True, axis = 0)
```

`This will delete all the rows which are containing the null values`

```
df.dropna(inplace = True, axis = 1)
```

`This will delete all the columns containing null values`

`i) Fillna Function`

Pandas Series.fillna() function is used to fill NA/NaN values using the specified method.

Suppose if we want to fill the null values with something instead of removing them then we can use fillna function Here we will be filling the numerical columns with its mean values and Categorical columns with its mode

```
link = 'https://raw.githubusercontent.com/AshishJangra27/Data-
Analysis-with-Python-GFG/main/3.%20Data%20Preprocessing%20-%20Removing
%20Null%20Value%20Rows/googleplaystore.csv'

df = pd.read_csv(link)

print(len(df))

10841
```

`Numerical columns`

```
mis = round(df['Rating'].mean(),2)

df['Rating'] = df['Rating'].fillna(mis)

print(len(df))
df.isna().sum()

10841

App                 0
Category            0
Rating              0
Reviews             0
Size                0
Installs            0
Type                1
Price               0
Content Rating      1
Genres              0
Last Updated        0
Current Ver         8
```

```
Android Ver       3
dtype: int64
```

If we would have used inplcae=True then it would have permenantly stored those values in our dataframe

`Categorical values`

```
df['Current Ver'] = df['Current Ver'].fillna('Varies on Device')
```

`j) sample function`

Pandas sample() is used to generate a sample random row or column from the function caller data frame.

Syntax:

DataFrame.sample(n=None, frac=None, replace=False, weights=None, random_state=None, axis=None)

```
df.sample(5)

                                 App           Category  Rating  
Reviews  \
28             Pencil Sketch Drawing     ART_AND_DESIGN    3.90  
136
6606                      BP Service           BUSINESS    4.19  
0
6740                    B@dL!bs Lite               GAME    3.80  
10
1638  Housing-Real Estate & Property          LIFESTYLE    4.10  
28301
4376    m.ride - your motorcycle app  AUTO_AND_VEHICLES    4.50  
189

                    Size    Installs  Type Price Content Rating  \
28                  4.6M     10,000+  Free     0       Everyone   
6606                 26M        100+  Free     0       Everyone   
6740                 36M      1,000+  Free     0     Mature 17+   
1638  Varies with device  1,000,000+  Free     0       Everyone   
4376                 16M     10,000+  Free     0       Everyone   

               Genres      Last Updated     Current Ver   Android Ver  

28       Art & Design     July 12, 2018             6.0    2.3 and up  

6606         Business  January 17, 2018  Rocksteady 1.3    4.1 and up  

6740             Word     April 5, 2016               6  4.0.3 and up  
```

```
1638         Lifestyle     July 13, 2018           12.1.0    4.1 and up

4376   Auto & Vehicles      July 3, 2018            1.1.6    6.0 and up
```

`k) to_csv() function`

Pandas Series.to_csv() function write the given series object to a comma-separated values (csv) file/format.

Syntax: Series.to_csv(*args, **kwargs)

```
data = { 'one'   : pd.Series([1, 2, 3, 4]),
         'two'   : pd.Series([10, 20, 30, 40]),
         'three' : pd.Series([100, 200, 300, 400]),
         'four'  : pd.Series([1000, 2000, 3000, 4000])}

df = pd.DataFrame(data)

df.to_csv('Number.csv')
```

- We got an extra Unnamed:0 Column if we want to avoid that we need to add an extra parameter mentioning index=False

```
df.to_csv('Numbers.csv', index = False)
```

4. A detailed Pandas Profile report

The pandas_profiling library in Python include a method named as ProfileReport() which generate a basic report on the input DataFrame.

The report consist of the following:

DataFrame overview, Each attribute on which DataFrame is defined, Correlations between attributes (Pearson Correlation and Spearman Correlation), and A sample of DataFrame.

```
from ydata_profiling import ProfileReport
import matplotlib.pyplot as plt
import pandas as pd


<IPython.core.display.HTML object>

# !pip install numpy==1.24.4 --force-reinstall
# # !pip install --upgrade pip setuptools wheel

# from ydata_profiling import ProfileReport

# import numpy as np
# print(np.__version__)
# !pip install --upgrade numpy scipy

# import ydata_profiling

df = pd.read_csv('Football.csv')
# df.head()

# report = pp.ProfileReport(df)
report = ProfileReport(df)

report
```

{"model_id":"ebf1bf0767604407b2ac5508631f0662","version_major":2,"version_minor":0}

```
  0%|
| 0/15 [00:00<?, ?it/s]
```

{"model_id":"816d853c6d5e45509de8dc5c2c9f0b5b","version_major":2,"version_minor":0}

{"model_id":"b7aadbee777a4a1fa2ff078a21c10e52","version_major":2,"version_minor":0}

```
<IPython.core.display.HTML object>
```